MotoShield: Open Miniaturized DC Motor Hardware Prototype for Control Education

Gergely Takács*, Ján Boldocký, Erik Mikuláš, Tibor Konkoly and Martin Gulan
Institute of Automation, Measurement and Applied Informatics, Faculty of Mechanical Engineering
Slovak University of Technology in Bratislava, Bratislava, Slovakia
*gergely.takacs@stuba.sk

***Abstract*—This article introduces a prototype reference design for the classical motor speed tracking experiment that is extensively used to teach control engineering and mechatronics concepts all over the globe. The literature is abundant with similar experimental setups combining a direct current motor and an incremental encoder, with the aim of controlling the rotational speed using feedback methods. The didactic device proposed here is special in that it integrates this setup into a miniaturized package at a very low cost, effectively creating take-home experimental systems suitable for remote teaching. In our concept, the motor-encoder coupled with power and measurement electronics is tightly integrated in a so-called shield, which is compatible with a range of microcontroller prototyping boards from the Arduino ecosystem and third-party compatibles. The hardware reference design is offered as an open-source, community driven initiative. In addition to the proposed device we present an application programming interface to abstract fundamental hardware functions and aid the education of higher concepts in control, system identification, signal processing or mechatronics. The programming interface is available in C/C++ for the Arduino IDE, in the MATLAB scripting language and as a Simulink blockset. Demonstration examples for feedback control and system identification are also provided for each environment. The combination of the open-source hardware and programming interface creates a powerful concept that is founded on standardization, meaning that educators may collaborate not only in bettering the existing hardware and software, but also on creating common teaching materials.**

***Index Terms*—open educational resources, control engineering education, microcontrollers, student experiments, mechatronics, educational technology, Arduino, DC motor**

I. Introduction

Future control systems engineers and mechatronics specialists pass through rigorous introductory courses, where they learn about the theoretical fundamentals of their trade in mathematical terms. These academic curricula are then enhanced by computer simulations of feedback control systems. However, it is, when students first experience laboratory trainers and other benchtop experiments that they first realize that the neat assumptions on nominal models, noise-free environments and linearity do not always suffice when it comes to misbehaving physical plants. Hands-on education enables students to implement control theory in lower-level compiled languages and forces them to factor in necessities such as analog-to-digital conversion (ADC) or timing for discrete sampling.

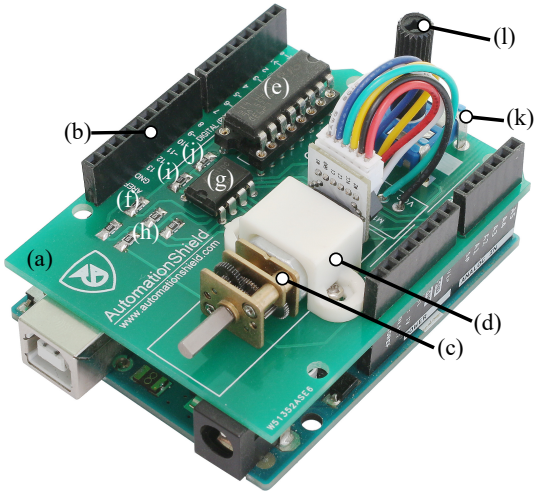


Fig. 1. Photograph of the MotoShield prototype.

There is an entire industry specialized in providing laboratory equipment for control engineering and mechatronics education and research. These professionally made precision instruments are identically made, come with technical support, and often contain training materials for instructors. On the other hand, the cost can range from several thousands up to tens of thousands of dollars for a single device. Thus, equipping a teaching or research laboratory for a group of students may be a costly endeavour [1]. Naturally, professional equipment may not be lent to students to take home for project work or assignments. In addition to the prohibitive price, borrowing hardware is also prevented by the size and fragility of the equipment [1] . The hardware and application-programming interfaces (API) tend to be closed-source proprietary products.

Another route to equipping teaching laboratories chosen by many resourceful instructors is to create benchtop trainers themselves; possibly integrating the task with the educational process itself and involving the students by thesis works and course projects. The cost of equipment may be minimal, but such improvised devices tend to

remain unique to the given laboratory and lack documentation. Improvisation in hardware is also a relative term, as on the extreme end of the spectrum, one may find LEGO blocks [2], [3], wood [4], salvaged components, and even cardboard [5]. Thus, replicating research results by others or creating universal—even open—courseware remains out of reach.

The financial burden of equipping research and teaching laboratories may be so severe that institutes of higher education on tight budgets, or countries with lower economic prowess may be excluded from adequate hands-on control engineering training. A quick literature research reveals numerous papers utilizing highly improvised hardware solutions replacing commercial laboratory systems (e.g. [4], [6]–[11]), and most works are tied to institutes located in countries that cannot replicate the economic might of world leaders. Inclusion in a competitive control engineering and mechatronics education then becomes not an issue of gender, race and similar factors; but rather a question of whether different can afford universities adequately equipped laboratories.

The question is, is there a middle ground in which equipment cost may be kept at absolute minimum, yet the hardware is replicable, well-designed and open to improvement by those interested to do so? In our previous work [12] we have laid out a philosophical framework to create open-source hardware for control engineering and mechatronics education that remains extremely low-cost, yet was designed with effortless replication in mind. The resulting devices are built as extension modules, or so-called "shields" to the popular Arduino microcontroller unit (MCU) prototyping ecosystem and thus, are small enough to fit in the palm of one's hand. These miniaturized take-home laboratories fit the scheme of so-called pocket laboratories and come with API and examples, mapping out the future path to open-source courseware as well. In our previous work we have proposed open hardware and corresponding software for the air levitation of a ball [12], the popular "ball on beam" experiment [13], magnetic levitation [14], rotational flexible beam experiment [15], a thermodynamic [16] and a simple optical experiment [17]. In this paper we take a closer look at an essential and common part of any control laboratory, the humble DC motor speed control.

Test rigs implementing DC or other types of electric motors for educational and research purposes are sold commercially. Examples include the Bytronic DC Motor Traing System, TecQuipment CE110, edibon SERIN series, FESTO 91024, Feedback 33-033 [18], Feedback DCM150F [19] or the now discontinued Quanser Engineering Trainer (QET) [20], QNET and QNET 2.0 DC Motor Boards [21].

Similarly, there is no shortage of academic articles utilizing various improvised DC speed control instruments. Most authors, especially in research articles, do not pay much attention to describe the utilized test equipment,

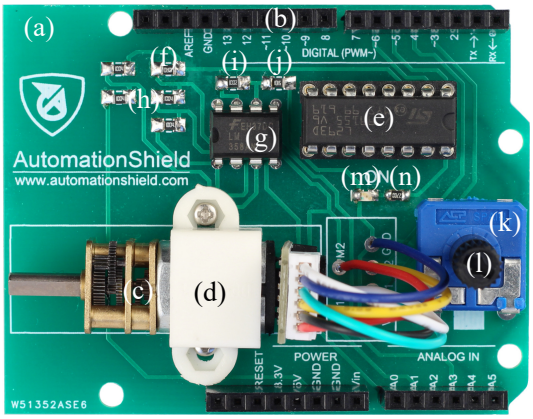


Fig. 2. Top view of the MotoShield device.

but renders experiments unrepeatable. A large industrial motor is considered in an improvised setting by Choudhary et al. in [22], nevertheless, such setups require safety training and are ill-advised for learning. A self-made smaller sized rig is considered in combination with the MyRIO proprietary hardware interface by Rata et al. in [23] and a programmable logic controller (PLC) in [24]. Čapková et al. feature a custom motor speed-control rig hosted in a 3D printed enclosure in their research [25]. Anishchenko and Zaleskyi propose a design within the pocket laboratory framework [4], however, the presented prototype is neither pocketable nor replicable as it is made of wood and salvaged components. Yfoulis et al. inches closer to a low-cost and small laboratory rig by combining the Arduino board with a micro motor equipped with an encoder [10], albeit realizing the hardware only on a breadboard. Recently Cabré et al. introduced a didactic platform for DC motor speed control using the Arduino Due and a custom external PCB containing power electronics [26]. Although they provide an excellent and extensive discussion of the possibilities in teaching control theory, the hardware is of limited utility to others.

Many others conducted and published similar research- and education-focused works combining the Arduino and DC motors (e.g [6]–[9], [11]), signifying that there is a great interest in carrying out such experiments. Nevertheless, these works have not created a reusable and universal experimental setting and comprehensive software support with examples, instead demonstrated improvised and temporary setups.

In this paper, we take a different approach and, based on earlier efforts (c.f. [27], [28]), propose a simple DC motor speed control device that is integrated to an Arduino shield (see Fig. 1 and Fig. 2). The hardware cost is on the order of ∼$20 (U.S.), and the design is entirely open-source. The device that we shall refer to as the "MotoShield" has an application programming interface in C/C++, MATLAB and Simulink and includes detailed instructional examples. Let us start with the hardware description.

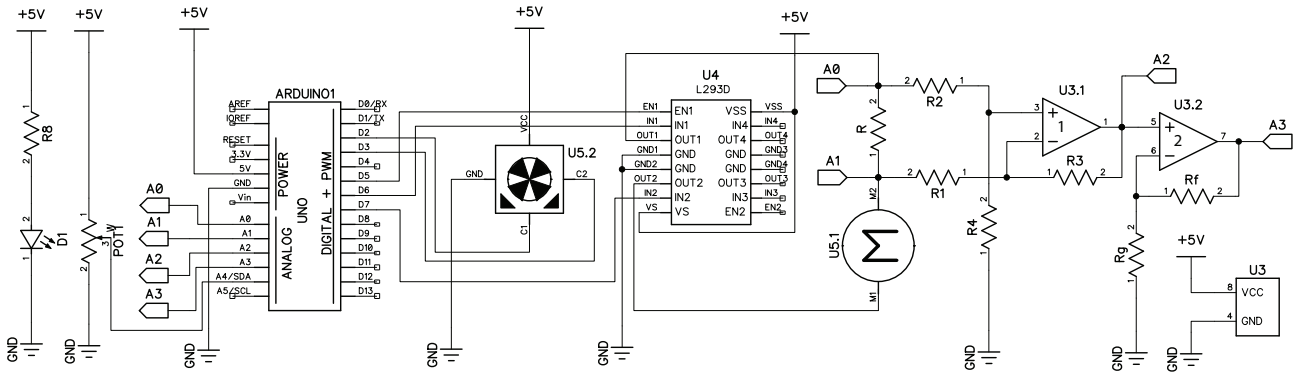


Fig. 3. Electrical schematics of the proposed device.

II. Hardware

The following passages will comprehensively describe the hardware design of the MotoShield prototype, so that potential replication and improvements may be realized effortlessly. The summary marks components by alphanumeric designators in parentheses that are repeated consistently accross the photographs of the assembled device in Figs. 1–2, the circuit schematics in Fig. 3 or the component list in Tab. I.

The entire experimental system rests on a printed circuit board (PCB) made from the customary glass-reinforced epoxy laminate (a). This board hosts the motor with the gearbox on its surface and connects to the microcontroller development board underneath electrically and mechanically by stackable header pins (b). The geometry of the PCB and the electrical role of the pins follows the specifications laid out for the Arduino R3 design and is also considered in similar compatible expansion shields. The PCB thus serves not only as a host for the electronic traces and components, but replaces a case or chassis for the system.

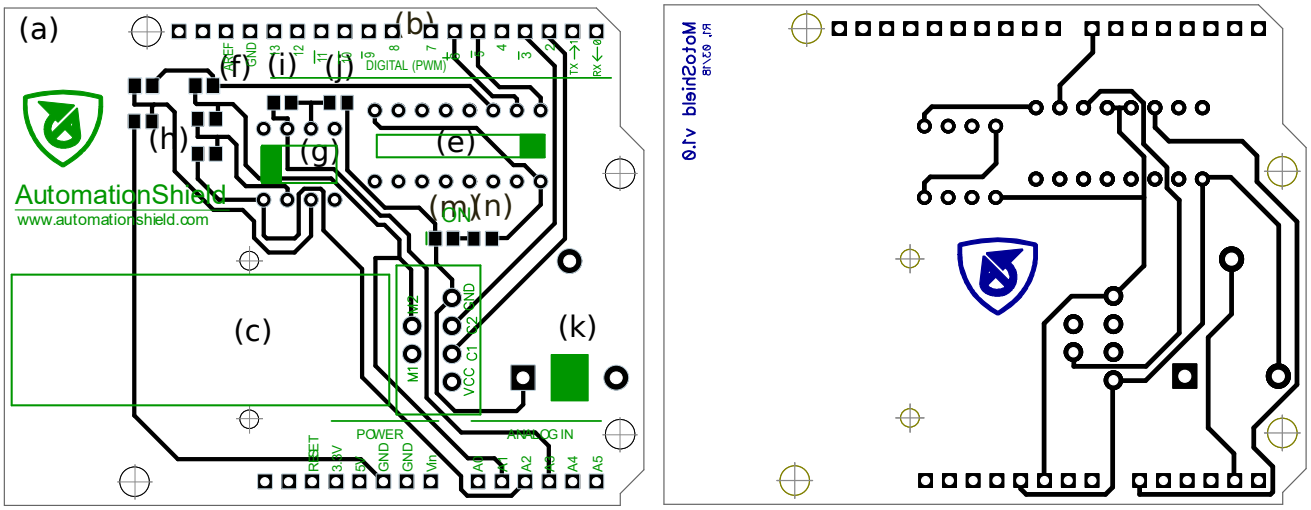
The heart of the experimental device is of course the direct-current (DC) brushed micro-motor in combination with a magnetic rotation encoder and a high, 380:1 ratio gearbox (c). This component may be purchased as a single integrated unit, rendering final assembly even easier. The 6 V motor draws 60 mA without load (170 mA stall), therefore, an external wall-plug adapter is not necessary and the USB power supply of the MCU is adequate in most cases. The motor is held in place by a plastic bracket (d) that can be readily purchased, or, alternatively, 3D printed. The bracket is bolted to the PCB by the holes drilled at the manufacturing stage.

The motor is driven by the L293 quadruple half H-bridge integrated circuit (e). Although better component choices may be made—especially considering that two half-bridges are not used—we have preferred the L293 because of universal acceptance and availability. The IC is enabled by the `D5` digital pin of the MCU, while the two half-bridges are controlled by the `D6` and `D7` GPIO pins. The chip draws power from the 5 V rail of the MCU-board and the outputs of the pair of half H-bridges are connected to the DC motor.

The speed of the shaft is sensed by the seven pole-pair magnetic Hall-effect encoder integrated to the back of the DC motor, producing $7\times2\times380$ pulses for one revolution of the output shaft. The resulting signals are passed to the `D2` and `D3` digital inputs of the MCU board.

Sensing the current consumed by the motor is not directly required to control its speed, however, it is a useful input to measure when creating models or designing state-feedback controllers. The current flowing to the motor passes through a shunt resistor (f) with a known nominal resistance of 10 Ω . The small voltage drop across the resistor must be first calculated, then amplified before it can be passed to the microcontroller. In our design this is carried out by the LM358 dual operational amplifier (g). The first operational amplifier is in a differential configuration with unit gain set by a cluster of identical resistors (h), the second takes the output and amplifies it by a gain that is configured by the pair of feedback (i) and gain (j) resistors. The amplified voltage drop can then be measured by the `A3` analog input of the microcontroller. The high- and low-end measurements around the shunt are also directly connected to the `A0` and `A1` analog inputs, while the output of the differential amplifier may be measured by the `A2` input of the microcontroller. These signals serve for diagnostic and educational purposes only; e.g. students may learn about the amplifiers by connecting an oscilloscope to these pins or measure signals by the MCU directly.

Finally, a potentiometer (k) with a corresponding plastic shaft (l) serves the user as a physical input interface to determine the reference speed to track by the controller algorithm, although its role is fully programmable as it is simply connected to the `A4` analog input of the MCU. Moreover, the “on” state of the microcontroller is represented by an onboard LED, which is covered and hidden by the shield. To replicate this essential function a LED (m) current-limited by a resistor (n) is added to the shield.



(a) Top layer.

(b) Bottom layer.

Fig. 4. Printed circuit board with a 1:1 scale of the proposed device showing traces and pads (black), drilled holes (gray) and the silkscreen layers (green, blue).

The electronic schematics, implementing the functional connections described above, is shown in Fig. 3. The circuit has been created in the free version of the DipTrace schematic and PCB design software, which can be used for this simple circuit practically without limitations. The circuit has been then converted to the two-layer PCB layout featured in Fig. 4. The PCB fits the customary 100 × 100 mm limit of most manufacturing services, allowing low-cost prototyping. As the MotoShield is an open-source hardware device, the editable schematics and PCB, along with the Gerber outputs format ready for instant manufacturing are available online [29], this way 5 pieces of PCB can be ordered instantly for as low as $2.

The bill of materials required to complete a MotoShield is listed in Tab. I, which shows component prices and a total estimate in U.S. dollars. The cost estimate is for low quantity component orders and excludes labor, postage, and consumables. The price of $22 can be realistically kept by individual students or hobbyists with access to basic bench equipment, while scaling production up, such as for an entire university course, will mean additional savings. We have not included the Arduino or compatible third-party prototyping board in this final price estimate, since the pricing varies and the device may be readily reused. All components have been selected with global ease of availability in mind. Although the circuit board contains surface mount devices (SMD), these are comfortably solderable even by beginners. One may think of assembling the device as an educational experience in itself.

The hardware design is only compatible to microcontrollers operating with TTL logic levels such as the AVR ATmega family, including the Arduino Uno, Arduino Mega and other third party variants.

III. Software

The hardware described above is only a part of the proposed didactic instrument. Interfacing the MotoShield with an intended feedback control application can be a valuable learning experience, however, it is time consuming and requires specialized knowledge in electronics and embedded systems. In order to expedite the learning process in the field of automation and mechatronics, the MotoShield comes with an open-source application programming interface. The role of the API is to create an abstraction layer between essential hardware functionality and the higher-level application itself. The API for the MotoShield is integrated into a larger collection of software called the AutomationShield Library[1], serving as a repository of API and examples for a range of similar devices.

A. Arduino API

The interface written in C/C++ is intended for use with the Arduino integrated development environment (IDE), and as such, has the structure of a standard Arduino Library. The library architecture is enforced by an automatic linting process across the repository, while all examples are subject to a build test as well.

From the perspective of the user, the MotoShield API is contained in the corresponding pair of `MotoShield.h` header and `MotoShield.cpp` implementation file. Methods defining the API are part of the `MotoShieldClass` that is initialized automatically as the `MotoShield` object. The naming scheme of methods conforms to syntactic Arduino customs, thus the hardware is simply initialized by the

```
MotoShield.begin(Ts);
```

[1]https://github.com/gergelytakacs/AutomationShield/

TABLE I
COMPONENT LIST OF THE PROPOSED DEVICE, WITHOUT AN ARDUINO BOARD[a,b].

Name	Description	PCB	Mark	Pcs.	Unit	Total ($)
PCB	FR4, 2 layer, 1.6 mm thick		(a)	1	0.48	0.48
DC motor	DFRobot FIT0487 DC motor with gearbox and encoder	U5	(c)	1	12.20	12.20
Motor bracket	DFRobot FIT0160 plastic bracket for motor with bolts and nuts		(d)	1	3.68	3.68
Opamp	LM358AN, operational amplifier	U3	(g)	1	0.24	0.24
Motor Driver	L293D, push-pull 4-channel motor driver	U4	(e)	1	3.59	3.59
DIP16 socket	DIP16 IC socket	U4		1	0.08	0.08
Resistor	1 MΩ 0.5%, 0805, SMD	R1,R2,R3,R4	(h)	4	0.03	0.12
Resistor	10 Ω 0.1%, 0805, SMD	R	(f)	1	0.03	0.03
Resistor	10 kΩ 0.5%, 0805, SMD	Rf	(i)	1	0.04	0.04
Resistor	5.1 kΩ 0.5%, 0805, SMD	Rg	(j)	1	0.016	0.016
Resistor	270 Ω 5%, 0805, SMD	R8	(m)	1	0.004	0.004
LED	Red LED, 0805, SMD	D1	(n)	1	0.07	0.07
Pot shaft	5×18.7 mm; e.g. ACP 14187-NE		(l)	1	0.10	0.10
Trimmer	10 kΩ , 250 mW, single turn THT trimmer	POT1	(k)	1	0.34	0.34
Header	10×1, female, stackable, 0.1“ pitch (e.g. SparkFun 474-PRT-10007)		(b)	1	0.072	0.072
Header	8×1, female, stackable, 0.1“ pitch (e.g. SparkFun 474-PRT-10007)		(b)	2	0.22	0.22
Header	6×1, female, stackable, 0.1“ pitch (e.g. SparkFun 474-PRT-10007)		(b)	1	0.11	0.11
					Total:	**$ 21.39**[a,b]

[a] For low quantity orders.
[b] Excluding labor and postage.

method, where T_s is the desired sampling period in milliseconds. The method sets the direction of general purpose input-output pins (GPIO), initializes the sampling subsystem of the AutomationShield Library, determines rotational directions, and configures the interrupts for counting the pulses originating in the encoder.

Shall the student wish to display variables inside the Arduino Serial Plotter built-in the IDE, the best practice is to scale everything to percentual values. For this the method

```
MotoShield.calibrate();
```

maps the lower and upper speed limits of the motor. The speed at full output is stored in `maxRPM` in revolutions per minute (RPM) units, while the minimal speed (`minRPM`) and the corresponding pulse width modulated (PWM) signal duty cycle is accessible via the `minDuty` variable.

This is a single-input and single-output (SISO) system and as such, there is a need to send inputs u_k to one actuator—the motor—via the

```
MotoShield.actuatorWrite(u);
```

method, accepting a floating-point variable `u` containing the desired duty cycle of the PWM signal (0—100%). The user sets the power sent to the motor through the power circuitry this way. Internally, the method checks for constraints to avoid overflow, maps the input to 8-bit PWM integers, then sends it to the `D3` pin of the Arduino. The equivalent no-load root mean square (RMS) input voltage may also be sent to the actuator by the `actuatorWriteVolt()` method.

Conversely, the only output y_k to read is the motor speed, which is accessible by invoking

```
y = MotoShield.sensorRead();
```

that will store the speed as a relative percentual value in a floating point variable `y`. Internally the routine reads the interrupt count, recomputes it to RPM, then scales it back to percents based on the calibration. More advanced users may alternatively register the speed directly in RPM by the `sensorReadRPM()` method.

To set the direction for the motor rotation, the

```
MotoShield.setDirection(dir);
```

method may be invoked. The boolean `dir` argument rotates the motor clockwise when true, which is the default option.

As we have noted earlier in the paper, state-feedback control and modeling may benefit from additional parameter readings. The current passing through the motor may be measured by the

```
i = MotoShield.sensorReadCurrent();
```

method, returning the current reading to the floating-point variable `i` in milliamperes. The routine reads the `A3` analog input and recalculates to voltage, then compensates for the gain of the non-inverting operational amplifier and finally enumerates the current based on the resistance of the shunt. The voltage drop on the shunt is directly accessible by calling `sensorReadVoltage()`.

Finally, the speed reference r_k to track by the controller may be set externally by turning the built-in potentiometer. The position of its runner is returned to a floating point variable `r` (0–100 %) by calling

```
r = MotoShield.referenceRead();
```

The AutomationShield Library contains numerous auxiliary methods and functions that are not specific to the hardware proposed here. These, amongst others, include strict real-time sampling subsystems for various microcontroller architectures, a proportional integral derivative (PID) controller, Kalman filter, running mean filter, a low-power interface (c.f. [30]) and others. The scope of this paper does not allow a further discussion of these features.

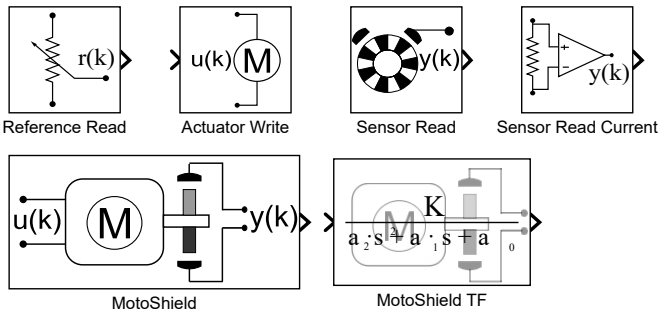


Fig. 5. Algorithmic blocks for the MotoShield API in Simulink.

B. MATLAB API

It is common to start learning control engineering concepts using MATLAB, since this high-level language implements numerous easy to use functions aimed at creating and simulating control applications. Although MATLAB is an interpreted language unfit for the direct implementation of control systems in practice, it is still possible to pair it with physical hardware and use it in an educational setting.

The MotoShield API requires the MATLAB Support Package for Arduino to communicate with the hardware. The API is deployed by the user launching an installation script once, then the methods contained within the `MotoShield` class in `MotoShield.m` are ready for operation.

The nomenclature of the Arduino API is also adopted for other environments, such as MATLAB. The interface remains largely identical across various implementations of the IDE, thus moving from, e.g., C/C++ to MATLAB is practically effortless for the user. For example, first an instance of the MotoShield class is created by

```
MotoShield = MotoShield;
```

command, then the hardware is initialized by calling the

```
MotoShield.begin('COM4','UNO');
```

method expecting the serial port and type of Arduino board as an input argument.

The rest of the functionality has the same syntax as explained above, thus `calibration()`, `sensorRead()`, `actuatorWrite()` or `referenceRead()` are fully compatible to the C/C++ version. Even though there are environment-specific changes to the MATLAB API, the internal logic of the routines also conforms to the one detailed above for the Arduino IDE, so it will not be detailed further here.

The hardware combined with the API may then be utilized for system identification and feedback control education or, even research. This means that high level and advanced control concepts may be conveniently tested and one can fully exploit the powerful environment MATLAB offers. The only caveat is sampling: this is only ensured by timing on the computer, the commands are transferred through the serial interface, thus sampling is not real-time in the strictest sense.

C. Simulink API

Similar to the MATLAB API, the MotoShield Simulink API requires the Simulink Support Package for Arduino Hardware. After running an installation script, the AutomationShield Simulink library will appear in the tree, offering various algorithmic blocks implementing the hardware interface.

The algorithmic blocks for the MotoShield are shown in Fig. 5. Individual blocks, such as `Actuator Write` or `Sensor Read` enable full control over the device, while the `MotoShield` block represents the system compactly as an input-output interface. After combining the API with various block libraries offered by Simulink, the user may deploy the application to the microcontroller board and run the entire test in the so-called "External" mode, allowing data logging and the incorporation of interactive elements such as sliders and switches.

All software interfaces have been tested using the Arduino Uno, other non-AVR designs such as the ARM Cortex M-based Due are not supported at the time because of logic-level voltage incompatibility.

IV. Examples

The AutomationShield Library contains a range of worked demonstration examples in addition to the hardware-specific API described above. The following passages illustrate potential topics for undergraduates, the possibilities are only limited by computation constraints (speed, memory).

A. System Identification

Let us assume that the motor can be described by a linear, time-invariant (LTI) system. Assuming $\ddot{\omega}(t)$ (rad·s^{-2}) is the angular acceleration of the shaft and the armature, furthermore, J (kg·m^2) is its rotational inertia, we have $J\ddot{\omega}(t) = M_\mathrm{d}(t) + M_\mathrm{f}(t)$. The moment $M_\mathrm{d}(t)$ (N·m) drives the motor proportional to applied current $i(t)$ (A), while $M_\mathrm{f}(t)$ (N·m) represents friction and damping effects and acts against the former with a moment proportional to motor speed $\dot{\omega}(t)$ (rad·s^{-1}). Assuming K_t is the motor driving moment coefficient and b (N·s·rad^{-1}·m^{-1}) is the viscous friction and damping coefficient we get

$$J\ddot{\omega}(t) \quad = \quad K_\mathrm{t} i(t) - b\dot{\omega}(t) \tag{1}$$

for the rotational dynamics.

For the electrical equation we shall turn to Kirchhoff's second law. Assuming the motor windings have a resistance of R (Ω) and inductance of L (H) and the voltage at its terminals is $U(t)$ (V) we have

$$U(t) \quad = \quad Ri(t) + L\frac{\mathrm{d}i(t)}{\mathrm{d}t} + \varepsilon(t), \tag{2}$$

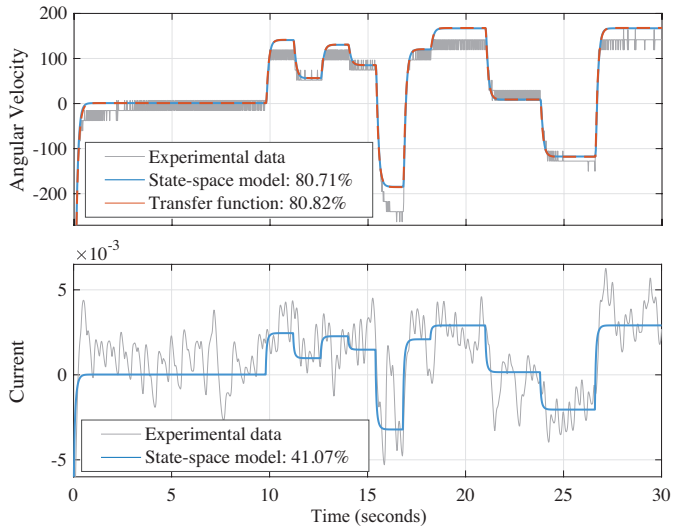


Fig. 6. Simulated response comparison of grey-box models: linear state-space model (blue) and linear transfer function (orange), including measurement data (grey).

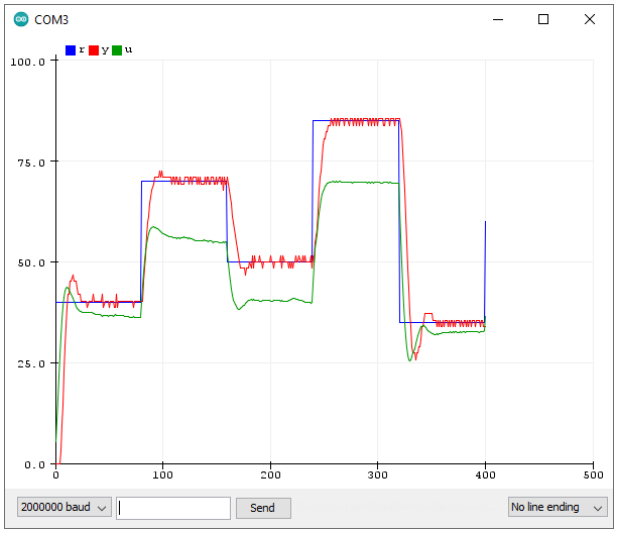


Fig. 7. Screenshot of closed-loop control in the Arduino IDE Serial Plotter (blue – reference, red – output, green – input). The vertical axis is motor speed (%) and motor PWM signal (%).

where $\varepsilon(t)$ (V) is the electromotive force proportional to motor speed by the electromotive coefficient K_e (V·s·rad^{-1}), yielding

$$U(t) \quad = \quad Ri(t) + L\frac{\mathrm{d}i(t)}{\mathrm{d}t} + K_\mathrm{e}\dot{\omega}(t). \tag{3}$$

After expressing the coefficients K_t and K_e in SI units we can state that $K = K_\mathrm{t} = K_\mathrm{e}$.

After performing the Laplace transformation on Eq. (1) and Eq. (3) students may derive a simple second-order continuous-time SISO transfer function as

$$G(s) = \frac{K}{K^2 + (Js+b)(Ls+R)}. \tag{4}$$

Alternatively, after choosing angular speed and current as state variables $\mathbf{x}(t) = \begin{bmatrix}\dot{\omega}(t) & i(t)\end{bmatrix}^\mathrm{T}$, a linear continuous state-space representation may be expressed as

$$\begin{bmatrix}\ddot{\omega}(t)\\ \dot{i}(t)\end{bmatrix} = \begin{bmatrix}-\frac{b}{J} & \frac{K}{J}\\ -\frac{K}{L} & -\frac{R}{L}\end{bmatrix}\begin{bmatrix}\dot{\omega}(t)\\ i(t)\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{L}\end{bmatrix}U(t). \tag{5}$$

The first-principle modeling procedure for the dynamics of the DC motor summarized above is well-known and possibly a part of every existing curriculum on control theory and identification. Students often proceed to simulation and controller design with parameters simply provided by the instructor. In our experience, many lack a practical understanding of designing identification experiments and extracting parameter estimates from data.

The AutomationShield Library contains several examples to illustrate this process. First, an amplitude modulated pseudo-random test signal is generated and exported to a C header (`MotoShield_APRBS.m`), then the experiment is run on the hardware (`MotoShield_Identification.ino`), logging the output speed and current into a file. The results are pre-processed, then a grey-box identification procedure is launched for the continuous-time transfer function (`MotoShield_GreyBox_ID_TF.m`) in Eq. (4) or the linear continuous state-space model (`MotoShield_GreyBox_ID_StateSpace.m`) in Eq. (5) using the MATLAB System Identification Toolbox. These worked examples include explanatory comments at every line and may be used with the supplied measurements as well. The results of this identification procedure are illustrated in Fig. 6 and provide an over 80% match to the measurement for the angular velocity.

B. Feedback Control

The first experience of undergraduate students with feedback control is certainly a variant of the PID control algorithm, usually in a simulation setting. Progressing from simulation to an actual embedded application is an important step in their professional development. The AutomationShield Library contains worked examples for PID control for all supported environments.

The example in the Arduino API (`MotoShield_PID.ino`)—similar to the other programming environments—implements a discrete PID algorithm in its absolute form, extended by input saturation and integral anti-windup by clamping. The controller is hand-tuned to $K_\mathrm{P} = 0.000001$, $T_\mathrm{I} = 0.0003$ and $T_\mathrm{D} = 0.001$, sampled at 40 ms. The measurements are fed to the serial port and may be conveniently displayed in the Arduino Serial Plotter (see Fig. 7) or a range of free third-party terminal applications. Note that variables are mapped to relative percentual values for easy readability of charts in the Serial Plotter, however, measurements in physical units are also feasible.

The same PID control application is faithfully reproduced in MATLAB, meaning that the code is written and

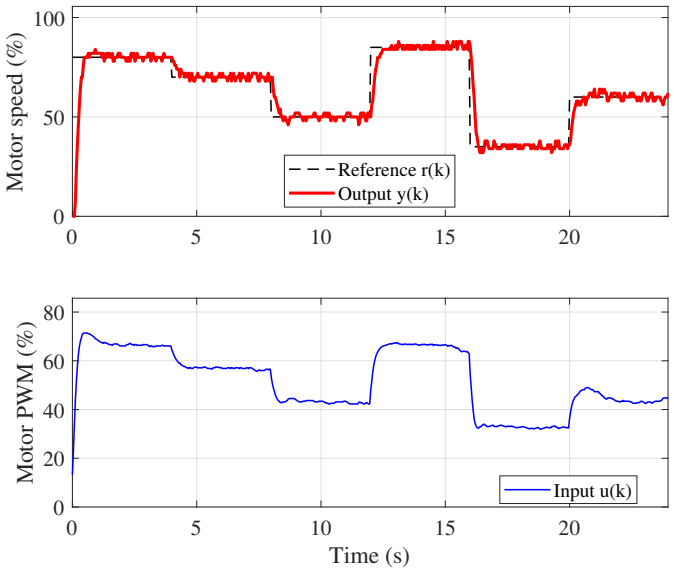


Fig. 8. PID control, created and executed in MATLAB.

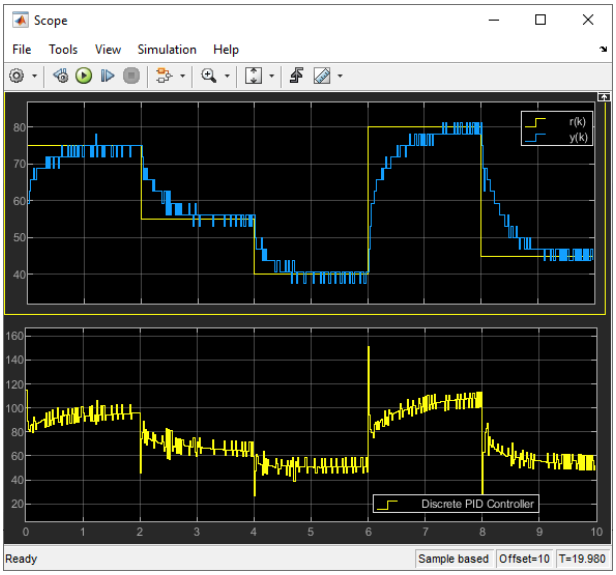


Fig. 10. Screenshot of live PID control on a Simulink Scope.

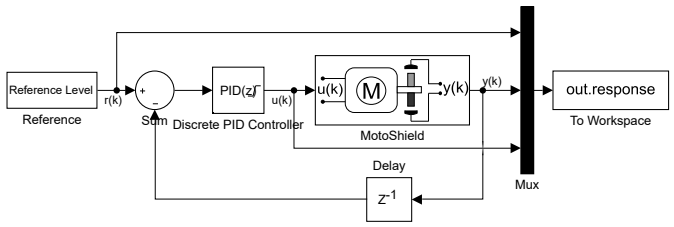


Fig. 9. PID control of the MotoShield in Simulink.

directly executed in this interpreted scripting language. Results can be conveniently stored, post-processed and displayed straightforwardly in MATLAB—see Fig. 8. As many students experience feedback control initially in MATLAB, testing theoretical concepts directly on hardware can be valuable. Although sampling is pseudo real-time, this has no visible effects on control quality given the manageable dynamics of the system.

Finally, Fig. 9 illustrates the ease of implementing feedback control for the MotoShield in Simulink. The scheme utilizes the `MotoShield` block, which solves interfacing with hardware. The rest of the algorithmic units, including the PID controller, are taken from the standard libraries provided by Simulink. Results may be logged to Workspace or displayed live on a virtual Scope, as it is demonstrated in Fig. 10.

V. Conclusion

This article proposed a miniaturized open-source version of the classical DC motor speed control experiment. The resulting hardware is a true pocket laboratory, which—thanks to its low-cost—is suitable for take-home experiments. Application programming interfaces are provided in C/C++ for the Arduino IDE, in MATLAB and for Simulink. These API abstract hardware functionality and enable students to work on higher-level control concepts instantly. In addition to the API we provide sample demonstration examples for system identification and feedback control.

A. Further work

A new version of the hardware is already under development. The power electronics will be thoroughly redesigned, including the substitution of the operational amplifier by a purpose-specific integrated circuit. The main improvement will be compatibility with CMOS logic level microcontrollers, thus, expanding the range of compatible prototyping boards.

The current API shall be augmented for compatibility with other microcontroller boards. The authors are also planning to extend the range of demonstration examples with more advanced control concepts, such as linear quadratic regulator and model predictive control. Additional application programming interfaces, such as in CircuitPython, Octave, LabView are also planned.

Acknowledgements

The authors gratefully acknowledge the contribution of the Slovak Research and Development Agency (APVV) under the contracts APVV-18-0023, APVV-17-0214 and APVV-14-0399. The authors appreciate the financial support provided by the Cultural and Educational Grant Agency (KEGA) of the Ministry of Education of Slovak Republic under the contract 012STU-4/2021.

References

[1] M. A. Hopkins and A. M. Kibbe, "Open-source hardware in controls education," in *Proceedings of the 121st ASEE Annual Conference & Exhibition*, Indianapolis, IN, USA, June 2014, pp. 24.955.1– 24.955.11, paper ID: 8778.

[2] A. Soriano, L. Marín, M. Vallés, A. Valera, and P. Albertos, "Low cost platform for automatic control education based on open hardware." *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 9044–9050, 2014, 19th IFAC World Congress.

[3] A. Tota, "DASL wiki: Ball and Beam v2," Online, 2016, [cited 04.20.2021]; Available from https://www.daslhub.org/unlv/wiki/doku.php?id=lego_ball_and_beam_v2. University of Nevada, Las Vegas (UNLV) Drones and Autonomous Systems Laboratory (DASL).

[4] M. Anishchenko and V. Zaleskyi, "Pocket labs development using ni mydaq data acquisition device," in *2020 IEEE Problems of Automated Electrodrive. Theory and Practice (PAEP)*, 2020, pp. 1–4.

[5] K. Benchikha, "Ball and Beam: PID control on Arduino with LabView to stabilize a ball on a beam," Online, 2019, [cited 04.20.2021]; Available from https://create.arduino.cc/projecthub/karem_benchikha/ball-and-beam-601d7a. Arduino Project Hub.

[6] A. Ma'arif, Iswanto, N. M. Raharja, P. Aditya Rosyady, A. R. Cahya Baswara, and A. Anggari Nuryono, "Control of dc motor using proportional integral derivative (pid): Arduino hardware implementation," in *2020 2nd International Conference on Industrial Electrical and Electronics (ICIEE)*, 2020, pp. 74–78.

[7] V. K. Singh, A. Sahu, A. Beg, B. Khan, and S. Kumar, "Speed direction control of dc motor through bluetooth hc-05 using arduino," in *2018 International Conference on Advanced Computation and Telecommunication (ICACAT)*, 2018, pp. 1–3.

[8] Z. Tir, O. Malik, M. A. Hamida, H. Cherif, Y. Bekakra, and A. Kadrine, "Implementation of a fuzzy logic speed controller for a permanent magnet dc motor using a low-cost arduino platform," in *2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B)*, 2017, pp. 1–4.

[9] Z. Adel, A. A. Hamou, and S. Abdellatif, "Design of real-time pid tracking controller using arduino mega 2560for a permanent magnet dc motor under real disturbances." in *2018 International Conference on Electrical Sciences and Technologies in Maghreb (CISTEM)*, 2018, pp. 1–5.

[10] C. Yfoulis, S. Papadopoulou, D. Trigkas, and S. Voutetakis, "Switching pi speed control of a nonlinear laboratory dc micromotor using low-cost embedded control hardware and software," in *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2018, pp. 1029–1034.

[11] S. Angalaeswari, A. Kumar, D. Kumar, and S. Bhadoriya, "Speed control of permanent magnet (PM)DC motor using Arduino and LabVIEW," in *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 2016, pp. 1–6.

[12] G. Takács, P. Chmurčiak, M. Gulan, E. Mikuláš, J. Kulhánek, G. Penzinger, M. Podbielančík, M. Lučan, P. Šálka, and D. Šroba, "FloatShield: An open source air levitation device for control engineering education," in *Proceedings of the 2020 IFAC World Congress*, Berlin, Germany, July 2020, pp. 1–8, (Preprints).

[13] G. Takács, E. Mikuláš, A. Vargová, T. Konkoly, P. Šíma, L. Vadovič, M. Bíro, M. Michal, M. Šimovec, and M. Gulan, "An Open-Source Miniature "Ball and Beam" Device for Control Engineering Education," in *Proceedings of the 2021 EDUCON IEEE Global Engineering Education Conference*, Vienna, Austria, April 2021, pp. –, (Preprints).

[14] G. Takács, J. Mihalík, E. Mikuláš, and M. Gulan, "MagnetoShield: Prototype of a low-cost magnetic levitation device for control education," in *Proceedings of the 2020 IEEE Global Engineering Education Conference (EDUCON)*, Porto, Portugal, April 2020, pp. 1516–1525.
on Sound and Vibration (ICSV), Prague, Czech Republic, July 2021, pp. 1–8, (Preprints).

[15] G. Takács, M. Vríčan, E. Mikuláš, and M. Gulan, "LinkShield: An early hardware prototype of a miniature low-cost flexible link experiment," in *Proceedings of the 27th International Congress*

[16] G. Takács, M. Gulan, J. Bavlna, R. Köplinger, M. Kováč, E. Mikuláš, S. Zarghoon, and R. Salíni, "HeatShield: a low-cost didactic device for control education simulating 3D printer heater blocks," in *Proceedings of the 2019 IEEE Global Engineering Education Conference*, Dubai, United Arab Emirates, April 2019, pp. 374–383.

[17] G. Takács, T. Konkoly, and M. Gulan, "Optoshield: A low-cost tool for control and mechatronics education," in *Proceedings of the 12th Asian Control Conference*, Kitakyushu-shi, Japan, Jun 2019, pp. 1001–1006.

[18] C. R. Figueroa, E. Rubio, I. Santana, J. Rohten, V. Esparza, and B. L. Martinez-Jimenez, "Speed and position control practices through the remote laboratory: Sld-ubb," in *2018 IEEE International Conference on Automation/XXIII Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, 2018, pp. 1–6.

[19] J. J. Fuertes, S. Alonso, A. Morán, M. A. Prada, S. García, and C. del Canto, "Virtual and remote laboratory of a dc motor," *IFAC Proceedings Volumes*, vol. 45, no. 11, pp. 288–293, 2012, 9th IFAC Symposium Advances in Control Education. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015376187

[20] J. Apkarian and K. Astrom, "A laptop servo for control education," *IEEE Control Systems Magazine*, vol. 24, no. 5, pp. 70–73, 2004.

[21] K. Alli, "A LabVIEW-based online DC servomechanism control experiments incorporating PID controller for students' laboratory," *The International Journal of Electrical Engineering & Education*, 2019.

[22] A. Choudhary, S. A. Singh, M. F. Malik, A. Kumar, M. K. Pathak, and V. Kumar, "Virtual lab: Remote access and speed control of dc motor using ward-leonard system," in *2012 IEEE International Conference on Technology Enhanced Education (ICTEE)*, 2012, pp. 1–7.

[23] G. Rata, C. Bejenar, and M. Rata, "A solution for studying the d.c. motor control using ni myrio-1900," in *2019 8th International Conference on Modern Power Systems (MPS)*, 2019, pp. 1–4.

[24] R. Bayindir, S. Vadi, and F. Goksucukur, "Implementation of a plc and opc-based dc motor control laboratory," in *4th International Conference on Power Engineering, Energy and Electrical Drives*, 2013, pp. 1151–1155.

[25] R. Čápková, A. Kozáková, M. Minár, and K. Ondrejička, "Robust QFT-based control of the DC motor laboratory model," in *2020 Cybernetics Informatics (K I)*, 2020, pp. 1–6.

[26] T. P. Cabré, A. S. Vela, M. T. Ribes, J. M. Blanc, J. R. Pablo, and F. C. Sancho, "Didactic platform for dc motor speed and position control in z-plane," *ISA Transactions*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019057821001038

[27] T. Konkoly, "Experimentálne moduly pre výučbu automatizácie (Experimental modules for automation education)," 2018, Bachelor's thesis. Slovak University of Technology in Bratislava, Bratislava, Slovakia. Supervisor: Gergely Takács. ID: SjF-13432-81384.

[28] J. Boldocký, "MotoShield:Vývoj programátorského rozhrania, identifikácia, riadenie a úprava didaktického prístroja na riadenie rýchlosti motorov (MotoShield: Developing an application programming interface, identification, control and redesign of a didactic device for motor speed control)," 2020, Bachelor's thesis. Slovak University of Technology in Bratislava, Bratislava, Slovakia. Supervisor: Gergely Takács. ID: SjF-13432-92875.

[29] M. Gulan, G. Takács, and T. Konkoly, "Motoshield," Online, 2021, [cited 19.04.2021]; GitHub Wiki page. Available from https://github.com/gergelytakacs/AutomationShield/wiki/MotoShield.

[30] G. Takács, E. Mikuláš, M. Vríčan, and M. Gulan, "Current-saving sampling for the embedded implementation of positive position feedback," in *Proceedings of the 49th International Congress and Exposition on Noise Control Engineering (InterNoise): Advances in Noise and Vibration Control Technology*, Seoul, South Korea, August 2020, pp. 537–547.